Finding read-once refutations in 2CNF formulas and variants - a parameterized perspective

K. Subramani,[1] **Piotr Wojciechowski**[1*]
[1]LDCSEE,
West Virginia University,
Morgantown, WV
k.subramani@mail.wvu.edu, pwojciec@mail.wvu.edu

Abstract

This paper is concerned with the design and analysis of *parameterized* algorithms for determining read-once refutations (RORs) in two classes of constraints, viz., unsatisfiable 2CNF formulas and unsatisfiable UCS$^+$s. These two problems are respectively known as the R2CNF and the RUCS+ problems. Previous work has established that both these problems are **NP-complete**; the current work establishes that they are **fixed-parameter tractable** for suitably chosen parameters. The problem of identifying refutations is an important one, especially in domains such as artificial intelligence and real-time systems, in which certification is an important aspect of computation. In this context, read-once refutations are of particular importance, since these refutations are by definition "short", i.e., at most linear in the size of the input.

1 Introduction

This paper is concerned with the design of *practical* algorithms for two problems, viz., the R2CNF problem and the RUCS+ problem. The R2CNF problem is concerned with finding a read-once refutation (ROR) in an unsatisfiable 2CNF formula, whereas the RUCS+ problem is concerned with finding a read-once refutation in an unsatisfiable UTVPI constraint system containing additional non-UTVPI constraints. Both these problems are known to be **NP-complete** (see Section 3). In this paper, we show that both these problems are in the complexity class **FPT** (fixed-parameter tractable), for suitably chosen natural parameters.

Both 2CNF Boolean formulas and UTPVI constraints find applications in a number of domains such as artificial intelligence and program verification. However, this paper is concerned with *certification*. It is well-known that making an algorithm certifying enhances **trust** in its output (Blum, Luby, and Rubinfeld 1993). Refutations are a form of certification for negative instances. In other words, if a 2CNF formula is unsatisfiable, then a refutation serves to "certify" its unsatisfiability. In general, refutations need not be "short", i.e., polynomial in the size of the input. Likewise, identifying a "natural" certificate for a problem is not always easy (McConnell et al. 2011).

*This research was supported in part by the Air-Force Office of Scientific Research through Grant FA9550-19-1-017.

An interesting line of research is therefore to focus on restricted classes of refutations, which are provably short. Read-once refutations are one such class of refutations (Iwama and Miyano 1995). This paper focuses on the read-once refutation problem for two classes of constraints, as mentioned above.

The principal contributions of this paper are as follows:

1. An **FPT** algorithm for finding a read-once resolution refutation of a 2-CNF formula parameterized by the length of the refutation. We also show that this algorithm can be modified to find the shortest read-once refutation in the input formula.
2. An **FPT** algorithm for finding a read-once linear refutation of a system of UTVPI constraints containing several non-UTVPI constraints parameterized by the number of non-UTVPI constraints.

The rest of this paper is organized as follows: In Section 2, we describe the problems under consideration. Section 3 motivates our work and discusses existing work in related problems. In section 4, we provide our **FPT** algorithm for 2-CNF formulas. Section 5 contains our algorithm for systems of UTVPI constraints containing several non-UTVPI constraints. Finally, in Section 6, we summarize our results and describe avenues for future research.

2 Statement of Problems

In this section, we describe the problems under consideration. We are concerned with read-once refutations in two different systems.

First, we examine 2-CNF formulas.

Definition 2.1. *A* 2**-CNF clause** *is a CNF clause with at most* 2 *literals.*

Example 2.1. The clause $(x_1 \vee \neg x_2)$ is a 2-CNF clause. However, $(x_1 \vee x_2 \vee \neg x_3)$ is not since it has 3 literals.

Definition 2.2. *A* 2**-CNF formula** *is a Boolean formula in which each clause is a* 2*-CNF clause.*

We are also concerned with UTVPI constraint systems.

Definition 2.3. *A constraint of the form* $a_i \cdot x_i + a_j \cdot x_j \leq b$ *where* $a_i, a_j \in \{-1, 0, 1\}$ *and* $b \in \mathbb{Z}$*, is called a* **unit two variable per inequality (UTVPI) constraint**.

Note that if only one of a_i or a_j in a UTVPI constraint is non-zero then the constraint is called an **absolute constraint**.

Example 2.2. The following are UTVPI constraints:

- $x_1 - x_2 \leq 3$.
- $x_2 + x_4 \leq 5$.
- $-x_3 - x_4 \leq 2$.

Definition 2.4. *A conjunction of UTVPI constraints is called a* **UTVPI Constraint System (UCS)**.

We are interested in a generalization of UCSs which allows for the inclusion of several constraints of the form $\sum_{i=1}^{n} a_i \cdot x_i \leq b$. We refer to these constraints as non-UTVPI constraints.

Definition 2.5. *A UCS_k^+ is a UCS with k non-UTVPI constraints.*

For both 2-CNF formulas and UCS_k^+s we are interested in the problem of finding read-once refutations. However, for each system, the inference rules used in those refutations are different.

For 2-CNF formulas we prove infeasibility by resolution refutation. In such a refutation, each new clause is derived by a resolution step.

Definition 2.6. *A* **resolution step** *derives a resolvent clause from two parent clauses. A resolution step which resolves $(\alpha \vee \beta)$ from parent clauses $(\alpha \vee x)$ and $(\neg x \vee \beta)$ is denoted as*

$$(\alpha \vee x) \wedge (\neg x \vee \beta) \mid\!\!\frac{1}{RES} (\alpha \vee \beta).$$

A sequence of resolution steps that proves the infeasibility of a CNF formula Φ (by deriving the empty clause $\sqcup$) is known as a **resolution refutation**. Such a refutation is denoted as $\Phi \mid\!\!\frac{}{RES} \sqcup$.

In UCS_k^+s we are interested in refutations which prove that the system has no rational solutions. Such a refutation is called a **linear refutation**. In such a refutation, each new constraint is derived by the addition rule.

Definition 2.7.

$$\textbf{ADD}: \frac{\sum_{i=1}^{n} a_i \cdot x_i \leq b_1 \qquad \sum_{i=1}^{n} a'_i \cdot x_i \leq b_2}{\sum_{i=1}^{n} (a_i + a'_i) \cdot x_i \leq b_1 + b_2} \tag{1}$$

We refer to Rule (1) as the **addition (ADD) rule**.

A sequence of applications of the ADD rule that proves the infeasibility of a UCS_k^+ **U** (by deriving the contradiction $0 \leq b$ where $b < 0$) is known as a **linear refutation**.

For both systems, we are interested in refutations in which each clause or constraint is used at most once. such a refutation is called a read-once refutation.

Definition 2.8. *A* **Read-Once** *refutation is a refutation in which each constraint (or clause) can be used in at most one inference. This applies to constraints present in the original system as well as those derived as a result of previous inferences.*

Note that if a constraint (or clause) can be re-derived from a different set input constraints, then the constraint can be reused.

We can now define the problems examined in this paper.

Definition 2.9. **2-CNF k-ROR:** *Given 2-CNF formula Φ and positive integer k, does Φ have a read-once resolution refutation that uses at most k clauses of Φ?*

In a read-once resolution refutation, each resolution step decreases the number of clauses by 1, since the parent clauses cannot be re-used. Thus, a read-once resolution refutation of a 2-CNF formula Φ that uses at most k clauses from Φ has at most $(k-1)$ resolution steps.

Definition 2.10. **UCS_k^+ ROR:** *Given a UCS_k^+ **U**, does **U** have a read-once refutation using the ADD rule?*

3 Motivation and Related work

In this section, we motivate our work and discuss related work in the literature.

2CNF formulas form the basis of several applications in logic (Papadimitriou 1994) and AI (Böckenhauer et al. 2010). It is well-known that 2CNF Satisfiability is in **P** (Aspvall, Plass, and Tarjan 1979). Likewise, Unit Two Variable Per Inequality (UTVPI) constraints occur in a wide variety of domains, including but not limited to program verification (Miné 2006), array bounds checking (Lahiri and Musuvathi 2005), and abstract interpretation (Singh et al. 2019). More recently, we demonstrated the existence of polynomial time certifying algorithms for linear feasibility (Subramani and Wojciechowski 2017) and integer feasibility (Subramani and Wojciechowski 2018) in UTVPI constraint systems. As mentioned before, the focus of this paper is not on algorithm design for satisfiability. Indeed our focus is on obtaining algorithms for "certification", i.e., refutation problems.

Read-once resolution as a proof system for clausal boolean formulas has been detailed in (Iwama 1997). By construction, read-once resolutions are succinct. If such a proof system is complete as well, then the complexity classes **NP** and **coNP** would coincide. (Iwama 1997) argues that this proof system is valuable, despite its **incompleteness**, since read-once proofs can be easily "visualized." One of the results in (Iwama 1997) is that asking if a 3CNF formula has a read-once refutation is **NP-complete**. We improved upon this result in (Kleine Büning, Wojciechowski, and Subramani 2018), wherein it was shown that the problem of read-once resolution existence is **NP-complete** even for 2CNF formulas, i.e. R2CNF is **NP-complete**.

In (Subramani 2009), it was shown that the ROR problem is solvable in polynomial time for difference constraint systems. This result was recently generalized to UTVPI constraint systems (Subramani and Wojciechowki 2019). We also investigated the existence of read-once refutations in Horn constraint systems and read-once unit resolution refutations in Horn constraint systems. We showed that these problems are **NP-complete** (Kleine Büning, Wojciechowski, and Subramani 2019b; 2019a). From these results, it follows that the RUCS+ problem is **NP-complete** as well.

4 An FPT algorithm 2-CNF k-ROR

In this section, we describe an **FPT** algorithm for finding a read-once resolution refutation of a 2-CNF formula Φ. This

algorithm is parameterized by the number of clauses from Φ used in the refutation.

First we provide a randomized algorithm for solving the 2-CNF m-ROR problem for a 2-CNF formula Φ with m clauses over n variables. This algorithm proceeds as follows:

1. For each variable x_i in Φ.
   (a) Create the sets Φ_i^+ and Φ_i^-.
   (b) For each clause $\phi_j \in \Phi$ uniformly and at random choose whether ϕ_j is added to the set Φ_i^+ or the set Φ_i^-.
   (c) If the clause (x_i) can be derived from Φ_i^+ and the clause $(\neg x_i)$ can be derived from Φ_i^-, then return **true**.
2. Return **false**.

Note that checking if (x_i) can be derived from Φ_i^+ can be done as follows:

1. Construct the implication graph $\mathbf{G}_i^+$ corresponding to Φ_i^+ using the construction in (Cormen et al. 2009).
2. Check if the node x_i^+ is reachable from the node x_i^- in $\mathbf{G}_i^+$.

Using the same technique, we can check if $(\neg x_i)$ can be derived from Φ_i^-.

This can be accomplished in $O(m)$ time. Thus, this entire procedure runs in $O(m \cdot n)$ time.

We now show that Φ has a read-once resolution refutation if the algorithm returns **true**. We also show that if Φ has a read-once refutation that uses k clauses from Φ, the the algorithm returns **true** with probability at least $\frac{1}{2^k}$.

Theorem* 4.1.** *If the randomized algorithm returns* **true, then Φ has a read-once resolution refutation.*

Proof. If the algorithm returns **true**, then for some x_i there exist set Φ_i^+ and Φ_i^- such that the clause (x_i) can be derived from Φ_i^+ and the clause $(\neg x_i)$ can be derived from Φ_i^-. If the clause (x_i) can be derived from Φ_i^+, then it can be derived using each clause at most once (Kleine Büning, Wojciechowski, and Subramani 2018). The same holds for deriving $(\neg x_i)$ from Φ_i^-.

Since Φ_i^+ and Φ_i^- are disjoint, these two derivations can be combined with the final resolution $(x_i) \wedge (\neg x_i) \mid\!\!\frac{1}{RES}\, \sqcup$ to obtain a read-once resolution refutation of Φ. □

***Theorem* 4.2.** *If Φ has a read-once resolution refutation that uses at most k clauses from Φ, then the randomized algorithm will return* **true** *with probability at least $\frac{1}{2^k}$.*

Proof. Let R be a read-once resolution refutation of Φ of that uses at most k clauses from Φ. Let $(x_i) \wedge (\neg x_i) \mid\!\!\frac{1}{RES}\, \sqcup$ be the last resolution step of R. Since R is a read-once resolution refutation of Φ, (x_i) and $(\neg x_i)$ must be derived from disjoint subsets of Φ. Let $R^+ \subseteq \Phi$ be the set of clauses used to derive (x_i) and let $R^- \subseteq \Phi$ be the set of clauses used to derive $(\neg x_i)$.

The randomized algorithm will find R if every clause in R^+ is added to the set Φ_i^+ and every clause in R^- is added to the set Φ_i^-. The probability of this happening is $\frac{1}{2^k}$. □

To obtain an **FPT** algorithm for solving the 2-CNF k-ROR problem we will de-randomize this algorithm. This de-randomization utilizes (m, k)-universal sets which are defined as follows:

Definition 4.1. *Let S be a set of size m. An (m, k)-universal set is a family $\mathbf{U}$ of subsets of S such that for any set $R \subseteq S$ of size k the family $\{A \cap R : A \in \mathbf{U}\}$ contains every subset of R.*

As in the proof of Theorem 4.2, let R be a read-once resolution refutation of Φ of that uses at most k clauses from Φ. Let $\mathbf{U}$ be an (m, k)-universal set for Φ. Then, for some $A \in \mathbf{U}$, we have that $A \cap R = R^+$. Thus, $R^+ \subseteq A$ and $R^- \subseteq \Phi \setminus A$. This means that the clause (x_i) can be derived from A and the clause $(\neg x_i)$ can be derived from $\Phi \setminus A$.

Note that we can construct an (m, k)-universal set for Φ of size $2^k \cdot k^{O(\log k)} \cdot \log m$ in $O(2^k \cdot k^{O(\log k)} \cdot m \cdot \log m)$ time (Naor, Schulman, and Srinivasan 1995).

Thus, given k, checking if Φ has a read-once resolution refutation using at most k clauses from Φ can be done as follows:

1. Construct an (m, k)-universal set $\mathbf{U}$ for Φ. Note that $\mathbf{U}$ contains $2^k \cdot k^{O(\log k)} \cdot \log m$ subsets of Φ. This can be done in time $O(2^k \cdot k^{O(\log k)} \cdot m \cdot \log m)$.
2. For each $A \in \mathbf{U}$ and each variable x_i if the clause (x_i) can be derived from A and the clause $(\neg x_i)$ can be derived from $\Phi \setminus A$ then Φ has a read-once refutation.

This algorithm runs in time $O(2^k \cdot k^{O(\log k)} \cdot n \cdot m \cdot \log m)$. Thus, this is an **FPT** algorithm for the 2-CNF k-ROR problem.

Note that this algorithm can be easily updated to find the shortest read-once refutation. Let R^* be the shortest read-once refutation of a 2-CNF formula Φ. If R^* uses at most k clauses of Φ, then for some x_i, R^* can be split into the clauses used to derive (x_i) and the clauses used to derive $(\neg x_i)$. Since R^* uses at most k clauses of Φ, then this split is found by some set A in the universal set $\mathbf{U}$. Thus, finding the shortest derivation of (x_i) and $(\neg x_i)$ will find R^*. Thus, to find the shortest read-once refutation of Φ we need to do the following:

1. Construct an (m, k)-universal set $\mathbf{U}$ for Φ. Note that $\mathbf{U}$ contains $2^k \cdot k^{O(\log k)} \cdot \log m$ subsets of Φ. This can be done in time $O(2^k \cdot k^{O(\log k)} \cdot m \cdot \log m)$.
2. For each $A \in \mathbf{U}$ and each variable x_i if the clause (x_i) can be derived from A and the clause $(\neg x_i)$ can be derived from $\Phi \setminus A$ then find the shortest derivation of (x_i) and the shortest derivation $(\neg x_i)$. These together form a read-once refutation of Φ.
3. Return the shortest refutation found by the algorithm.

5 An FPT algorithm for UCS_k^+ ROR

In this section, we describe an **FPT** algorithm for finding a read-once linear refutation of a system of UTVPI constraints containing several non-UTVPI constraints. This algorithm is parameterized by the number of non-UTVPI constraints.

Let $\mathbf{U}$ be a UCS_k^+ with m constraints over n variables. We will use k as the parameter for our algorithm.

This algorithm reduces the problem to the problem of finding a minimum weight perfect matching (MWPM) in an undirected, weighted graph. This problem is defined as follows:

Definition 5.1. *Let $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{w} \rangle$ denote an undirected graph, with vertex set $\mathbf{V}$, edge set $\mathbf{E}$ and edge cost function $\mathbf{w}$. Let $n = |\mathbf{V}|$ and let $m = |\mathbf{E}|$. A* **matching** *is any collection of vertex-disjoint edges. A* **perfect matching** *is a matching in which each vertex $v \in \mathbf{V}$ is matched.*

The fastest strongly polynomial combinatorial algorithm for the MWPM problem runs in time $O(m \cdot n + n^2 \cdot \log n)$ (Gabow 1990). This is the bound that we use in our paper.

For each variable x_i let d_i be the total number of constraints in $\mathbf{U}$ that use the variable x_i. Additionally let d_0 denote the number of absolute constraints rounded up to the nearest even number. Also let $\mathbf{U_U}$ denote the set of UTVPI constraints in $\mathbf{U}$.

From $\mathbf{U_U}$, we can construct an undirected graph $\mathbf{G}$ as follows:

1. For each variable x_i, create the vertices $x^+_{i,1}, \ldots, x^+_{i,d_i}$ and $x^-_{i,1}, \ldots, x^-_{i,d_i}$. Also create the edges $x^+_{i,1} \overset{0}{---} x^-_{i,1}, \ldots, x^+_{i,d_i} \overset{0}{---} x^-_{i,d_i}$.
2. Create the vertices $x_{0,1}, \ldots, x_{0,d_0}$. Also create the edges $x_{0,1} \overset{0}{---} x_{0,2}$,$x_{0,3} \overset{0}{---} x_{0,4}, \ldots, x_{0,d_0-1} \overset{0}{---} x^-_{0,d_0}$.
3. For each constraint l_h in $\mathbf{U_U}$, create the vertices y_h and y'_h. Also create the edge $y_h \overset{0}{---} y'_h$.
   (a) If l_h is of the form $x_i + x_j \leq b_h$, then create the edges $x^+_{i,1} \overset{\frac{b_h}{2}}{---} y_h, \ldots, x^+_{i,d_i} \overset{\frac{b_h}{2}}{---} y_h$ and $x^+_{j,1} \overset{\frac{b_h}{2}}{---} y'_h, \ldots, x^+_{j,d_j} \overset{\frac{b_h}{2}}{---} y'_h$.
   (b) If l_h is of the form $x_i - x_j \leq b_h$, then create the edges $x^+_{i,1} \overset{\frac{b_h}{2}}{---} y_h, \ldots, x^+_{i,d_i} \overset{\frac{b_h}{2}}{---} y_h$ and $x^-_{j,1} \overset{\frac{b_h}{2}}{---} y'_h, \ldots, x^-_{j,d_j} \overset{\frac{b_h}{2}}{---} y'_h$.
   (c) If l_h is of the form $-x_i - x_j \leq b_h$, then create the edges $x^-_{i,1} \overset{\frac{b_h}{2}}{---} y_h, \ldots, x^-_{i,d_i} \overset{\frac{b_h}{2}}{---} y_h$ and $x^-_{j,1} \overset{\frac{b_h}{2}}{---} y'_h, \ldots, x^-_{j,d_j} \overset{\frac{b_h}{2}}{---} y'_h$.
   (d) If l_h is of the form $x_i \leq b_h$, then create the edges $x^+_{i,1} \overset{\frac{b_h}{2}}{---} y_h, \ldots, x^+_{i,d_i} \overset{\frac{b_h}{2}}{---} y_h$ and $x_{0,1} \overset{\frac{b_h}{2}}{---} y'_h, \ldots, x_{0,d_0} \overset{\frac{b_h}{2}}{---} y'_h$.
   (e) If l_h is of the form $-x_i \leq b_h$, then create the edges $x^-_{i,1} \overset{\frac{b_h}{2}}{---} y_h, \ldots, x^-_{i,d_i} \overset{\frac{b_h}{2}}{---} y_h$ and $x_{0,1} \overset{\frac{b_h}{2}}{---} y'_h, \ldots, x_{0,d_0} \overset{\frac{b_h}{2}}{---} y'_h$.

Example 5.1. Let us consider UCS (2).

$$\begin{array}{lrcl} l_1: & -x_1 + x_2 & \leq & -2 \\ l_2: & x_1 + x_3 & \leq & -2 \\ l_3: & -x_2 - x_3 & \leq & 2 \end{array} \tag{2}$$

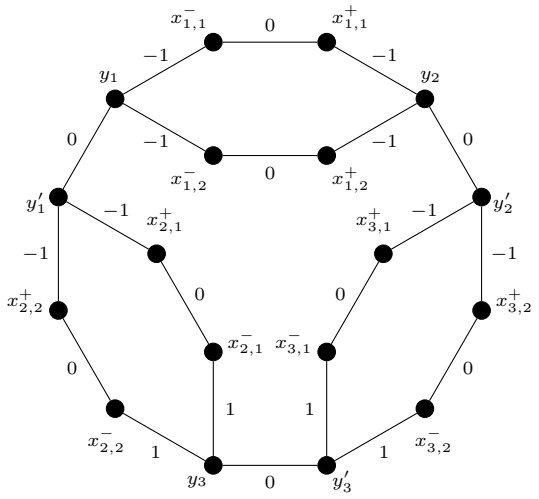

Figure 1: Undirected Graph Corresponding to UCS (2).

Applying the construction discussed above to UCS (2), we get the undirected graph in Figure 1.

This construction is a modified version of the graph construction used in (Subramani and Wojciechowki 2019). It differs in the following ways:

1. For each variable, x_i, we now have d_i pairs of vertices associated with that variable instead of 2. We will be utilizing this construction to derive more general constraints thus we may need to use each variable more than twice.
2. We no longer have a singe pair of vertices to handle absolute constraints. Instead, we have d_0 pairs. This allows the refutation to use all d_0 absolute constraints if necessary.
3. Despite these changes $\mathbf{G}$ still has a negative weight perfect matching if and only if $\mathbf{U_U}$ has a read-once refutation using only the ADD rule (Theorem 5.1).

We now relate read-once refutations of $\mathbf{U_U}$ to negative weight perfect matchings of the corresponding graph $\mathbf{G}$. This is broken up into two parts:

1. Using a read-once refutation of $\mathbf{U_U}$ to construct a negative weight perfect matching of $\mathbf{G}$.
2. Using a negative weight perfect matching of $\mathbf{G}$ to construct a read-once refutation of $\mathbf{U_U}$.

***Theorem* 5.1.** *$\mathbf{G}$ has a negative weight perfect matching if and only if $\mathbf{U_U}$ has a read-once refutation using only the ADD rule.*

Proof. First assume that $\mathbf{U_U}$ has a read-once refutation R.

We construct a negative weight perfect matching P of $\mathbf{G}$ as follows:

1. For each variable x_i:
   (a) Let c_i be the number of constraints from $\mathbf{U_U}$ in R that contain the term x_i. Note that this is also the number of constraints that contain the term $-x_i$.
   (b) Add the edges $x^+_{i,c_i+1} \overset{0}{---} x^-_{i,c_i+1}, \ldots, x^+_{i,d_i} \overset{0}{---} x^-_{i,d_i}$ to P.

2. Assume that the constraints in $\mathbf{U_U}$ are assigned an arbitrary order. For each constraint l_h in $\mathbf{U_U}$:
   (a) If $l_h \notin R$, add the edge $y_h \overset{0}{\text{---}} y'_h$ to P.
   (b) If $l_h \in R$ is of the form $a_i \cdot x_i + a_j \cdot x_j \leq b_h$ such that $a_i, a_j \neq 0$:
      i. If l_h is the rth constraint to use the term x_i, add the edge $x^+_{i,r} \overset{\frac{b_h}{2}}{\text{---}} y_h$ to P.
      ii. If l_h is the rth constraint to use the term $-x_i$, add the edge $x^-_{i,r} \overset{\frac{b_h}{2}}{\text{---}} y_h$ to P.
      iii. If l_h is the rth constraint to use the term x_j, add the edge $x^+_{j,r} \overset{\frac{b_h}{2}}{\text{---}} y_h$ to P.
      iv. If l_h is the rth constraint to use the term $-x_j$, add the edge $x^-_{j,r} \overset{\frac{b_h}{2}}{\text{---}} y_h$ to P.
   (c) If $l_k \in R$ is of the form $a_i \cdot x_i \leq b_k$:
      i. If l_h is the rth constraint to use the term x_i, add the edge $x^+_{i,r} \overset{\frac{b_h}{2}}{\text{---}} y_h$ to P.
      ii. If l_h is the rth constraint to use the term $-x_i$, add the edge $x^-_{i,r} \overset{\frac{b_h}{2}}{\text{---}} y_h$ to P.
      iii. If l_h is the rth absolute constraint, then add the edge $x_{0,r} \overset{\frac{b_h}{2}}{\text{---}} y'_h$ to P.
   (d) If R has c_0 absolute constraints, then add the edges $x_{0,c_0+1} \overset{0}{\text{---}} x_{0,c_0+2}$ through $x_{0,d_0-1} \overset{0}{\text{---}} x_{0,d_0}$ to P.

We now make the following observations:

1. For each variable x_i:
   (a) The vertices x^+_{i,c_i+1} through x^+_{i,d_i} and x^-_{i,c_i+1} through x^-_{i,d_i} are the end points of the edges $x^+_{i,c_i+1} \overset{0}{\text{---}} x^-_{i,c_i+1}$ through $x^+_{i,d_i} \overset{0}{\text{---}} x^-_{i,d_i}$ which are in P.
   (b) For each $r = 1 \ldots c_i$, the vertex $x^+_{i,r}$ is the endpoint of the edge $x^+_{i,r} \overset{b_h}{\text{---}} y_h$ or $x^+_{i,r} \overset{b_h}{\text{---}} y'_h$ where l_h is the rth constraint in R with the term x_i.
   (c) For each $r = 1 \ldots c_i$, the vertex $x^-_{i,r}$ is the endpoint of the edge $x^-_{i,r} \overset{b_h}{\text{---}} y_h$ or $x^-_{i,r} \overset{b_h}{\text{---}} y'_h$ where l_h is the rth constraint in R with the term $-x_i$.
2. For each constraint l_k:
   (a) If $l_k \in R$, then, by construction, P contains two edges of weight $\frac{b_k}{2}$ one with end point y_k and one with endpoint y'_k. Thus both l_k and l'_k are the endpoints of exactly one edge in P.
   (b) If $l_k \notin R$, then, by construction, P contains the edge $y_k \overset{0}{\text{---}} y'_k$ and none of the weight $\frac{b_k}{2}$ edges. Thus both y_k and y'_k are the endpoints of exactly one edge in P.
3. The vertices x_{0,c_0+1} through x_{0,d_0} are the end points of the edges $x_{0,c_0+1} \overset{0}{\text{---}} x_{0,c_0+2}$ through $x_{0,d_0-1} \overset{0}{\text{---}} x_{0,d_0}$ which are in P.
4. For each $r = 1 \ldots c_0$, the vertex $x_{0,r}$ is the endpoint of the edge $x_{0,r} \overset{b_h}{\text{---}} y'_h$ where l_h is the rth absolute constraint in R.

Thus, every vertex in $\mathbf{G}$ is an endpoint of exactly one edge in P. It follows that P is a perfect matching. Additionally, for each constraint $l_k \in R$, P has two edges of weight $\frac{b_k}{2}$ and all other edge in P have weight 0. Thus,

$$\sum_{e \in P} \mathbf{c}'(e) = \sum_{l_k \in R} \left(\frac{b_k}{2} + \frac{b_k}{2} \right) = \sum_{l_k \in R} b_k < 0.$$

This means that P has negative weight.

Now assume that $\mathbf{G}$ has a negative weight perfect matching P. We construct a read-once refutation R as follows:

1. For each constraint l_h in $\mathbf{U}$, if P does not use the edge $y_h \overset{0}{\text{---}} y'_h$, then add the constraint l_h to R.

P is a perfect matching. Thus, for each variable x_i, we have the following:

1. If a constraint $l_h \in R$ contains the term x_i, then for some r, the edge $y_h \overset{\frac{b_h}{2}}{\text{---}} x^+_{i,r}$ or $y'_h \overset{\frac{b_h}{2}}{\text{---}} x^+_{i,r}$ is in P. Thus, the edge $x^+_{i,r} \overset{0}{\text{---}} x^-_{i,r}$ is not in P. This means that for some h', one of the edges $y_{h'} \overset{\frac{b_{h'}}{2}}{\text{---}} x^-_{i,r}$ or $y'_{h'} \overset{\frac{b_{h'}}{2}}{\text{---}} x^-_{i,r}$ is in P. Thus, the term x_i is used contained in the constraint $l_{h'} \in R$. Note that each constraint in R that contains the term x_i corresponds to a different constraint in R that contains the term $-x_i$.
2. Similarly, each constraint in R that contains the term $-x_i$ corresponds to a different constraint in R that contains the term x_i.

Thus, the term x_i appears in the same number of constraints in R as the term $-x_i$.

If a constraint $l_k \in R$, then the edge $y_k \overset{0}{\text{---}} y'_k$ is not in P. Thus, P contains two edges of weight $\frac{b_k}{2}$ one with end point y_k and one with endpoint y'_k. Conversely, if the constraint $l_k \notin R$, then the edge $y_k \overset{0}{\text{---}} y'_k$ is in P. It follows that none of the edges of weight $\frac{b_j}{2}$ from y_j and y'_j are in P. Thus, for each constraint $l_k \in R$, P has two edges of weight $\frac{b_k}{2}$ and all other edge in P have weight 0

Thus, summing the constraints in R yields

$$0 \leq \sum_{l_j \in R} b_j = \sum_{l_j \in R} \left(\frac{b_j}{2} + \frac{b_j}{2} \right) = \sum_{e \in P} \mathbf{c}(e) < 0.$$

By construction, each constraint appears at most once in R. Thus, R is a read- once refutation of $\mathbf{U}$. □

The number of vertices in $\mathbf{G}$ can be counted as follows:

1. The constraints in $\mathbf{U}$ contribute $O(m)$ vertices to $\mathbf{G}$.
2. The variable x_i contributes $2 \cdot d_i$ vertices to $\mathbf{G}$.

3. In total the variables contribute $\sum_{i=1}^{n} 2 \cdot d_i$ vertices to $\mathbf{G}$. Since $\mathbf{G}$ is made from the UTVPI constraints in $\mathbf{U}$, we have that this value is in $O(m)$.
4. $\mathbf{G}$ has $O(m)$ total vertices.

The number of edges in $\mathbf{G}$ can be counted as follows:

1. For each variable x_i, there are d_i edges between the vertices corresponding to x_i. This contributes a total of $O(m)$ edges.
2. For each variable x_i, there are at most d_i^2 edges between the variable and constraint vertices. This contributes a total of $\sum_{i=1}^{n} d_i^2 \leq n \cdot \sum_{i=1}^{n} d_i$. Thus, this contributes $O(n \cdot m)$ edges.
3. Each constraint contributes a constant number of additional edges, 1 for non-absolute constraints and 4 for absolute constraints. Thus this accounts for $O(m)$ additional edges.
4. $\mathbf{G}$ has $O(n \cdot m)$ total edges.

Thus, finding the MWPM of $\mathbf{G}$ takes $O(m^2 \cdot (n + \log m))$ time.

We need to modify $\mathbf{G}$ to account for the non-UTVPI constraints in $\mathbf{U}$. Let S be an arbitrary subset of $\mathbf{U} \setminus \mathbf{U_U}$. Let l_S be the constraint derived by summing all of the constraints in S. This constraint is of the form $\sum_{i=1}^{n} a_{i,S} \cdot x_i \leq b_S$. To obtain a read-once refutation, we want to derive the constraint $\sum_{i=1}^{n} -a_{i,S} \cdot x_i \leq -b$ where $b > b_S$ from $\mathbf{U_U}$.

We now modify $\mathbf{G}$ to account for the constraint l_S. Let $\mathbf{G}_S$ be the modified graph. Initially $\mathbf{G}_S = \mathbf{G}$. We modify $\mathbf{G_S}$ as follows:

1. For each x_i:
   (a) If $a_{i,S} > 0$, then remove the vertices $x_{i,1}^{+}, \ldots, x_{i,a_{i,S}}^{+}$ from $\mathbf{G}_S$. This forces a read-once derivation corresponding to a perfect matching of $\mathbf{G}_S$ to use the term $-x_i$ exactly $a_{i,S}$ more times than it used x_i. Thus the coefficient of x_i in the derived constraint is $-a_{i,S}$ as desired.
   (b) If $a_{i,S} < 0$, then remove the vertices $x_{i,1}^{-}, \ldots, x_{i,|a_{i,S}|}^{-}$ from $\mathbf{G}_S$. This forces a read-once derivation corresponding to a perfect matching of $\mathbf{G}_S$ to use the term x_i exactly $|a_{i,S}|$ more times than it uses $-x_i$. Thus the coefficient of x_i in the derived constraint is $|a_{i,S}| = -a_{i,S}$ as desired.
2. If $\sum_{i=1}^{n} a_{i,S}$ is odd, then remove the vertex $x_{0,1}$ from $\mathbf{G}_S$. This forces the refutation to use an odd number of absolute constraints. It also ensures that an even number of vertices have been removed from $\mathbf{G}_S$.
3. Add the vertices y_S^{+} and y_S^{-} to $\mathbf{G}_S$ also add the edge $y_S^{+} \xrightarrow{b_S} y_S^{-}$ to $\mathbf{G}_S$. Note that this edge must be in any perfect matching. Thus, the remaining edges in a negative weight perfect matching must have total weight $-b$ where $b > b_S$.

From these observations and arguments analogous to those made in the proof of Theorem 5.1, we have the following corollary.

***Corollary* 5.1.** *For a UCS$_k^+$ $\mathbf{U}$ and set $S \subseteq \mathbf{U} \setminus \mathbf{U_U}$, $\mathbf{U}$ has a read-once refutation using all of the constraints in S if and only if the graph $\mathbf{G}_S$ has a negative weight perfect matching.*

We can now make an **FPT** algorithm for finding a read-once refutation of $\mathbf{U}$.

1. Construct the graph $\mathbf{G}$ from $\mathbf{U_U}$.
2. For each $S \subseteq \mathbf{U} \setminus \mathbf{U_U}$ (there are 2^k such subsets).
   (a) Modify $\mathbf{G}$ to construct $\mathbf{G}_S$.
   (b) Find the minimum weight perfect matching P of $\mathbf{G_S}$.
   (c) If P has negative weight, return that $\mathbf{U}$ has a read-once refutation.
3. If no S resulted in a $\mathbf{G}_S$ with a negative weight perfect matching, then return that $\mathbf{U}$ has no read-once refutation.

Since finding the MWPM of $\mathbf{G}$ takes $O(m^2 \cdot (n + \log m))$ time, this algorithm runs in $O(2^k \cdot m^2 \cdot (n + \log m))$ time. Thus, this is an **FPT** algorithm for the UCS$_k^+$ ROR problem.

6 Conclusion

In this paper, we constructed **FPT** algorithms for the following problems:

1. The 2-CNF k-ROR problem when parameterized by k, the number of clauses used by the refutation.
2. The UCS$_k^+$ ROR problem when parameterized by k, the number of non-UTVPI constraints.

We are interested in the following problems from the perspective of future research:

1. Without a value for k, the algorithm in this paper for determining if a 2-CNF formula Φ has a read-once refutation would need to check all 2^m possible partitionings of Φ into Φ_i^+ and Φ_i^-. This results in an $O^*(2^m)$ time exponential algorithm. Is there a more efficient way to do this?
2. Using the **FPT** algorithm in this paper, we can solve the UCS$_k^+$ ROR problem in polynomial time for any fixed k. The problem of finding a read-once refutation of a general linear program is **NP-hard** (Kleine Büning, Wojciechowski, and Subramani 2019a). If we allow k to change depending on the size of the UCS, for what values of k does this problem remain solvable in polynomial time?

References

Aspvall, B.; Plass, M. F.; and Tarjan, R. 1979. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters* 8(3):121–123.

Blum, M.; Luby, M.; and Rubinfeld, R. 1993. Self-testing/correcting with applications to numerical problems. *Journal of Computing and System Sciences* 47(3):549–595.

Böckenhauer, H.; Forisek, M.; Oravec, J.; Steffen, B.; Steinhöfel, K.; and Steinová, M. 2010. The uniform minimum-ones 2sat problem and its application to haplotype classification. *RAIRO - Theor. Inf. and Applic.* 44(3):363–377.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2009. *Introduction to Algorithms*. Cambridge, MA: The MIT Press, 3rd edition.

Gabow, H. N. 1990. Data structures for weighted matching and nearest common ancestors with linking. In Johnson, D., ed., *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, 434–443. San Francisco, CA, USA: SIAM.

Iwama, K., and Miyano, E. 1995. Intractability of read-once resolution. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory (SCTC '95)*, 29–36. Los Alamitos, CA, USA: IEEE Computer Society Press.

Iwama, K. 1997. Complexity of finding short resolution proofs. *Lecture Notes in Computer Science* 1295:309–319.

Kleine Büning, H.; Wojciechowski, P. J.; and Subramani, K. 2018. Finding read-once resolution refutations in systems of 2cnf clauses. *Theor. Comput. Sci.* 729:42–56.

Kleine Büning, H.; Wojciechowski, P. J.; and Subramani, K. 2019a. On the application of restricted cutting plane systems to horn constraint systems. In *The 12th International Symposium on Frontiers of Combining Systems, London, United Kingdom,, September 4-6, 2019, Proceedings*, 149–164.

Kleine Büning, H.; Wojciechowski, P. J.; and Subramani, K. 2019b. Read-once resolutions in Horn formulas. In *Frontiers in Algorithmics - 13th International Workshop, FAW 2019, Sanya, China, April 29 - May 3, 2019, Proceedings*, 100–110.

Lahiri, S. K., and Musuvathi, M. 2005. An Efficient Decision Procedure for UTVPI Constraints. In *Proceedings of the 5^{th} International Workshop on the Frontiers of Combining Systems, September 19-21, Vienna, Austria*, 168–183. New York: Springer.

McConnell, R. M.; Mehlhorn, K.; Näher, S.; and Schweitzer, P. 2011. Certifying algorithms. *Computer Science Review* 5(2):119–161.

Miné, A. 2006. The octagon abstract domain. *Higher-Order and Symbolic Computation* 19(1):31–100.

Naor, M.; Schulman, L. J.; and Srinivasan, A. 1995. Splitters and near-optimal derandomization. In *Proceedings of IEEE 36th Annual Foundations of Computer Science*, 182–191.

Papadimitriou, C. H. 1994. *Computational Complexity*. New York: Addison-Wesley.

Singh, G.; Gehr, T.; Püschel, M.; and Vechev, M. T. 2019. An abstract domain for certifying neural networks. *PACMPL* 3(POPL):41:1–41:30.

Subramani, K., and Wojciechowki, P. 2019. A polynomial time algorithm for read-once certification of linear infeasibility in UTVPI constraints. *Algorithmica* 81(7):2765–2794.

Subramani, K., and Wojciechowski, P. J. 2017. A combinatorial certifying algorithm for linear feasibility in UTVPI constraints. *Algorithmica* 78(1):166–208.

Subramani, K., and Wojciechowski, P. J. 2018. A certifying algorithm for lattice point feasibility in a system of utvpi constraints. *Journal of Combinatorial Optimization* 35(2):389–408.

Subramani, K. 2009. Optimal length resolution refutations of difference constraint systems. *Journal of Automated Reasoning (JAR)* 43(2):121–137.